Representation and Integration: Combining Robot Control, High-Level Planning, and Action Learning

Petrick, R., Kraft, D., Mourao, K., Geib, C., Pugeault, N., Krüger, N., & Steedman, M. (2008). Representation and Integration: Combining Robot Control, High-Level Planning, and Action Learning. In Proceedings of the International Cognitive Robotics Workshop (CogRob 2008) at ECAI 2008. (pp. 32-41).

Representation and Integration: Combining Robot Control, High-Level Planning, and Action Learning

Ronald Petrick[1], **Dirk Kraft**[2], **Kira Mourão**[1], **Christopher Geib**[1], **Nicolas Pugeault**[1,2], **Norbert Krüger**[2], and **Mark Steedman**[1]

Abstract. We describe an approach to integrated robot control, high-level planning, and action effect learning that attempts to overcome the representational difficulties that exist between these diverse areas. Our approach combines ideas from robot vision, knowledge-level planning, and connectionist machine learning, and focuses on the representational needs of these components. We also make use of a simple representational unit called an instantiated state transition fragment (ISTF) and a related structure called an object-action complex (OAC). The goal of this work is a general approach for inducing high-level action specifications, suitable for planning, from a robot's interactions with the world. We present a detailed overview of our approach and show how it supports the learning of certain aspects of a high-level representation from low-level world state information.

1 INTRODUCTION AND MOTIVATION

The problem of integrating low-level robot systems with high-level symbolic planners introduces significant representational difficulties that must first be overcome. Since the requirements for robot-level control and vision tend to be different from that of traditional planning, neither representation is usually sufficient to accommodate the needs of an integrated system. Overcoming these representational differences is a necessary challenge, however, since both levels seem to be required to produce human-like behaviour.

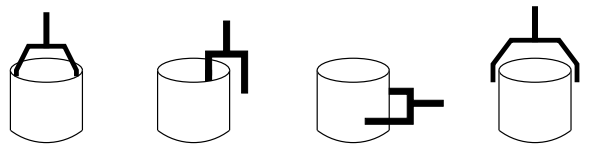
In general, robot systems tend to use representations based on vectors of continuous values, which denote 3D spatial coordinates, joint angles, force vectors, or other world-level features that require real-valued parameters [20]. Such representations allow system builders to succinctly specify robot behaviour since most of the computations required for low-level robot control are effectively captured as continuous transforms of continuous vectors over time. On the other hand, high-level planning systems typically use representations based on discrete, symbolic models of objects, properties, and actions, described in logical languages (e.g., [5, 23, 16, 27, 31]). Instead of modelling low-level continuous processes, these representations capture the dynamics of the world or the agent's knowledge at a more abstract level, for instance by characterizing the state changes that result from deliberate, planned action.

In this paper we describe an approach for integrating a robot/vision system with a high-level planner, that attempts to overcome the representational challenges described above. In particular, our approach gives rise to a system that is capable of automatically inducing certain aspects of a high-level representation suitable for planning, from the robot's interactions with the real world using basic "reflex" actions. This paper describes work currently in progress. As such, we do not address the entire problem of learning action representations, but instead focus on two important parts: object learning and action effect learning. Our approach uses a simple representational unit called an *instantiated state transition fragment (ISTF)* and a related structure called an *object-action complex (OAC)* [7], both of which arise naturally from the robot's interaction with the world—and world objects in particular. These notions also help us address certain control problems, for instance the relationship between high-level sensing actions and their execution by the robot, and representational issues that arise at different levels of our system. Although we only consider a portion of a larger learning problem, we are also interested in implementing these ideas within a framework that includes the lowest-level control mechanisms right up to the high-level reasoning components. Finally, we believe our approach is general and that these ideas can be successfully transferred to other robot platforms and planners, with capabilities other than those we describe here.

To illustrate our approach, we will consider a simple robot manipulation scenario throughout this paper. This domain consists of a robot with a gripper, a table with a number of objects on it, and a "shelf" (a special region of the table). The robot has a camera to view the objects in the world but does not initially have knowledge of those objects. Instead, world knowledge must be provided by the vision system, the robot's sensors, and the basic action reflexes built into the robot controller. The robot is given the task of clearing the objects from the table by placing them on the shelf. The shelf has limited space so the objects must be stacked in order for the robot to successfully complete the task. For simplicity, each object is assumed to be roughly cylindrical in shape and has a "radius" which provides an estimate of its size. An object A can be stacked into an object B provided the radius of A is less than that of B, and B is an "open" object. Unlike a standard blocks-world scenario, the robot will not have complete information about the state of the world. In particular, we will often consider scenarios where the robot does not know whether an object is open or not and must perform a test to determine an object's "openness". The robot will also have a choice of four different grasping types for manipulating objects in the world. Not all grasp types can be used on every object, and certain grasp types are further restricted by the position of an object relative to other objects in the world. The set of available grasp types is shown in Figure 1. Finally, actions in this domain can fail during execution and the robot's sensors may return noisy data.

The rest of the paper presents an overview of our approach from a representational point of view, and discusses the main components

[1] School of Informatics, University of Edinburgh,Edinburgh EH8 9LW, Scotland, United Kingdom, contact e-mail: *rpetrick@inf.ed.ac.uk*.
[2] The Maersk Mc-Kinney Moller Institute, University of Southern Denmark, DK-5230 Odense M, Denmark.



(a) Grasp A (b) Grasp B (c) Grasp C (d) Grasp D

Figure 1. Available grasping types in the robot manipulation scenario

of our system. In Section 2 we describe the basic representations used in the paper. In Section 3 we discuss how object information is discovered from the robot/vision system's initial experiences in the world. In Section 4 we describe the high-level planner and plan execution monitor. In Section 5 we introduce a mechanism for learning the effects of actions from state descriptions. In Section 6 we discuss the current state of implementation in our system and some early empirical results. Finally, in Section 7 we discuss the advantages of our approach from a representation point of view, and describe some areas of future work.

2 BASIC REPRESENTATIONS

At the robot/vision level, the system has a set Σ of sensors, $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$, where each sensor σ_i returns an observation $obs(\sigma_i)$ about some aspect of the world. The execution of a robot-level *motor program* or robot action may cause changes to the world which can be observed through subsequent sensing. Each motor program is typically executed with respect to particular *objects* in the world. We assume that the robot/vision system does not initially know about any objects and, thus, can't execute many motor programs. Instead, the robot has a set of *basic reflex actions* that aren't dependent on particular objects and can be used for exploring the world initially.

The planning level representation is based on a set of *fluents*, $f_1, f_2, \ldots, f_m$: first-order predicates and functions that denote particular qualities of the world, robot, and objects. Fluents represent high-level (possibly abstract) counterparts of some of the properties the robot is capable of sensing. In particular, the value of a fluent is a function of the observations returned by the sensor set, i.e., $f_i = \Gamma_i(\Sigma)$. Typically, each fluent depends on a subset of the sensor observations and not every sensor need map to a fluent (some sensors are only used at the lower control level). Fluents can also be parameterized by high-level versions of the objects known at the robot level.

A *state* is a snapshot of the values of all instantiated fluents at some point during the execution of the system. States represent a point of intersection between the low-level and high-level representations, since states are *induced* from a set of sensor observations and the corresponding sensor/fluent mappings (i.e., the functions Γ_i). High-level *actions* represent abstract versions of some of the robot's motor programs. Since all actions must ultimately be executed by the robot, each action is decomposable to a fixed set of motor programs.[3] Thus, the robot's interaction with the world can be viewed as a simple state transition system: the robot's sensor observations give rise to a state description; executing an action brings about changes in the world that can be observed through subsequent sensing. More importantly, every interaction of this form provides the robot with an opportunity to observe a small portion of the world's state space, which we refer to as an *instantiated state transition fragment (ISTF)* [7].

Formally, an ISTF is a tuple $\langle s_i, mp_j, obj_{mp_j}, s_{i+1} \rangle$, where s_i is the state that is sensed before applying the motor program instance mp_j, obj_{mp_j} is the object that the motor program is defined relative to, and s_{i+1} is the state sensed after executing the motor program. Thus, an ISTF is a situated pairing of an object and an action that captures a small fragment of the world's state transition function. The states s_i and s_{i+1} contain snapshots of the robot's information about these states, some of which may be irrelevant to the action being applied.

We will also consider a second representational structure that results from generalising over instances of ISTFs. Such structures are referred to as *object-action complexes (OACs)* [7], and are similar to ISTFs, but contain only the relevant instantiated state information needed to predict the applicability of an action and its effects, with all irrelevant information pruned away. An OAC is defined by a tuple of the form $\langle S_i, MP_j, Obj_k, S_{i+1} \rangle$, where S_i and S_{i+1} are two states, MP_j is a set of motor programs, and Obj_k is a class of objects. In this case, S_i only describes those properties of the world state that are required to execute any of the motor programs in MP_j, when acting on an object of class Obj_k. S_{i+1} describes a world state which captures the properties changed by the motor program.

Typically, we consider ISTFs and OACs formed from *partial* state descriptions. Such descriptions arise since the robot is not always able to sense the status of all objects and properties of the world (e.g., occluded or undiscovered objects). We also note that the robot's sensors may be noisy and, thus, there is no guarantee that state reports only contain correct information. Furthermore, certain sensor operations have associated resource costs (e.g., time, energy, etc.). For instance, the robot can perform a test to determine whether an object is open by "poking" the object to check its concavity. Such operations are only initiated on demand at the discretion of the high-level planning system.

3 VISION-BASED OBJECT DISCOVERY

The robot system includes a vision component based on an *early cognitive vision* framework [15] which provides a scene representation composed of local 3D edge descriptors that outline the visible contours [26]. Because the system initially lacks knowledge of the objects that make up the scene, the visual representation is *unsegmented*: descriptors that belong to one object are not explicitly distinct from the ones that belong to other objects, or the background.

To aid in the discovery of new objects, the robot is equipped with a basic reflex action [1] that is elicited by specific visual feature combinations in the unsegmented world representation. The outcome of these reflexes allows the system to gather knowledge about the scene, which is used to segment the visual world into objects and identify basic affordances. We consider a reflex where the robot tries to grasp a planar surface in the scene. Each time the robot executes such a reflex, haptic information allows the system to evaluate the outcome: either the grasp was successful and the gripper is holding something, or it failed and the gripper simply closed. A failed attempt forces the system to reconsider its original assumptions, whereas a successful attempt confirms the feasibility of the reflex. Once a successful grasp is performed, the robot gains physical control over this part of the scene. If we assume that the full kinematics of the robot's arm are known (which is true for industrial robots), then it is possible to segment the grasped object from the rest of the visual world as it is the only part that moves synchronously with the robot's arm.

With physical control, the system visually inspects an object from

[3] We do not focus here on the problem of learning high-level action *schema* (i.e., the set of action names and their parameters) or the action/motor program mappings. Instead, we assume that the action schema are provided with the corresponding mappings to robot-level motor programs.

a variety of viewpoints and builds a 3D representation [13]. Features on the object are tracked over multiple frames, between which the object moves with a known motion. If features are constant over a series of frames they become included in the object's representation, otherwise they are assumed not to belong to the object. (A more detailed description of the accumulation process can be found in [13].) The final description is labelled and recorded as an identifier for a new object class, along with the successful reflex (now a motor program). Using this new knowledge, the system then reconsiders its interpretation of the scene: using a representation-specific pose estimation algorithm [3] all other instances of the same object class are identified and labelled. By repeating this process, the system constructs a representation of the world objects, as instances of symbolic classes that carry basic affordances, i.e., particular reflex actions that have been successfully applied to grasp objects of this class.

The technical implementation of the pose estimation algorithm has only recently become available. Prior to this, a circle detection algorithm was developed to recognise cylindrical objects, to which the domain was restricted for this work. Four grasp templates were used to define the primitive reflex actions in an object-centric way (where concrete grasps were generated based on the object pose). Although this approach negates the need for the general pose estimation algorithm, the conclusions drawn from experiments in this limited scenario are still easily transferable to the general case.

Figure 2 illustrates the "birth of an object." In (a), the dots on the image show the predicted structures. Both spurious primitives, parts of the background that are not confirmed by the image, and the confirmed predictions are shown. In (b), the shape model learned from the object in (a) is shown. In (c) and (d), two additional objects are shown along with their learned shape models. The "gap" in the shape models corresponds to where the robot's gripper held the objects.

The object-centric nature of the robot's world exploration process has immediate consequences for the high-level representation. First, newly discovered objects are reported to the planning level and added to its representation. At the planning level, objects are simply labels while the real-world object information is stored at the robot level. Such a representation means that we can avoid certain types of real-valued information at the high level (e.g., 3D location coordinates, object orientation vectors, etc.) and instead refer to objects by their labels (e.g., *obj1* may denote a particular red cup on the table). With the addition of new objects, the planning system can immediately start using such objects in its reasoning and plan construction. Since we assume that object names do not change over time, high-level object labels act as indices into the low-level object representation. Thus, plans with object references will be understandable to the robot/vision system. Finally, the successful identification of new objects will cause the robot/vision system to start sending regular state updates to the planning level about these objects and their properties. In particular, the ISTFs that result from subsequent interactions with the world will contain state information about these objects, provided they can be sensed by the robot. The planning level can then use this information for plan construction and plan execution monitoring. Additional details about the link between the robot/vision and planning systems are given in Section 6.

4 PLAN GENERATION AND MONITORING

The high-level planner is responsible for constructing plans that direct the behaviour of the robot in order to achieve a set of goals. For instance, in our example domain a plan might be constructed to clear all "open" objects from the table. Plans are built using PKS ("Planning with Knowledge and Sensing") [24, 25], a knowledge-level conditional planner that can operate with incomplete information and high-level sensing actions. Like other symbolic planners, PKS requires a goal, a description of the initial state, and a list of the available actions before it can construct plans. Unlike traditional approaches, PKS operates at the *knowledge level* by modelling the agent's knowledge state, rather than the world state. By doing so, PKS can reason efficiently about certain types of knowledge, and make effective use of non-propositional features, like functions, which often arise in real-world scenarios.

PKS is based on a generalization of STRIPS [5]. In STRIPS, a single database represents the world state. Actions update the database in a way that corresponds to their effects on the world. In PKS, the planner's knowledge state is represented by five databases, each of which stores a particular type of knowledge. Actions are described by the changes they make to the database set and, thus, to the planner's knowledge state. PKS also supports ADL-style conditional action effects [23].

Using PKS's representation language, we can formally model the example robot scenario by describing the objects, properties, and actions that make up the planning level domain. As we described above, objects at the planning level are simply labels that denote actual objects in the world identified by the robot/vision system.

High-level domain properties are defined by sets of logical fluents, i.e., predicates and functions that denote particular qualities of the world, robot, and objects. For instance, to model the example object manipulation scenario we include fluents such as:

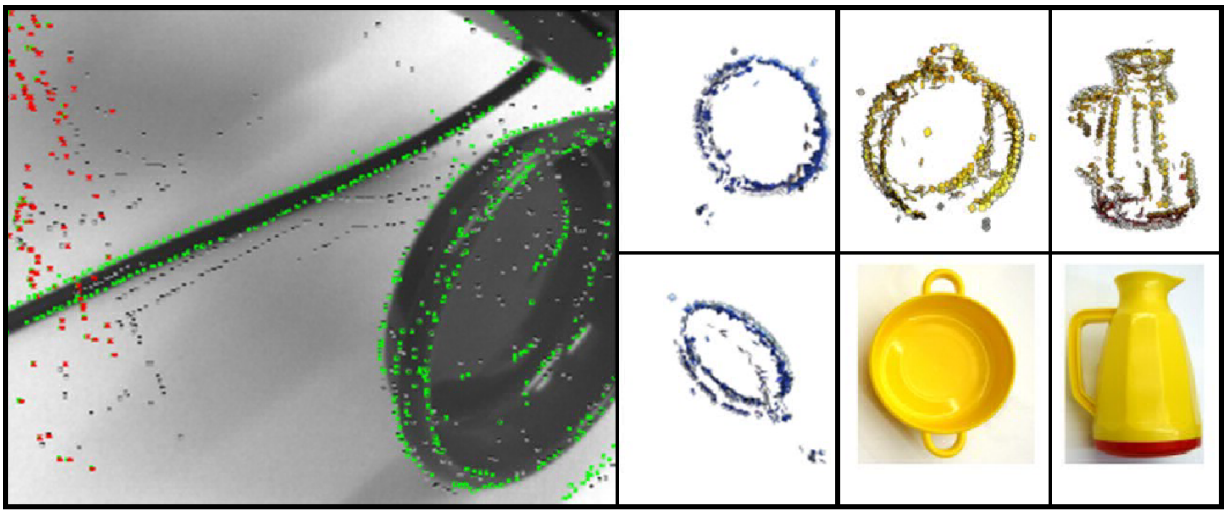
- *open*(x): object x is open,
- *gripperempty*: the robot's gripper is empty,
- *ingripper*(x): object x is in the gripper,
- *ontable*(x): object x is on the table,
- *isin*(x, y): object x is stacked in object y,
- *reachableX*(x): object x is reachable using grasp type X, and
- *radius*(x) = y: the radius of object x is y,

among others. While most high-level properties tend to abstract the information returned by a set of sensors at the robot level, some properties correspond more closely to individual sensors (e.g., *gripperempty* closely models a low-level sensor that detects whether the robot's gripper can be closed without contact, while *ontable* requires data from a set of visual sensors concerning object positions).

High-level actions represent counterparts to some of the motor programs available at the robot level. For instance, in the example scenario the planner has access to actions like:

- *graspA-stack*(x): grasp object x from a stack using grasp type "A",
- *graspA-table*(x): grasp x from the table using grasp A,
- *putInto-object*(x, y): put object x into an object y on the table,
- *putAway*(x): put x away on the shelf, and
- *findout-open*(x): determine whether x is open or not,

among others. Some actions like "grasp A" are divided into two actions to account for different object configurations, however, the motor programs that implement these actions do not necessarily make such distinctions. Furthermore, the object-centric nature of the planning actions means that they do not require 3D coordinates, joint angles, or similar real values but, instead, include parameters that can be instantiated with specific objects. Actions also exist for other grasping options (B, C, and D) available at the robot level. Actions like *findout-open* are high-level *sensing actions* that direct the robot to gather information about the world state that is not normally provided to the planner as part of its regular state updates.



(a) (b) (c) (d)

Figure 2. Birth of the object

Table 1. Examples of PKS actions in the object manipulation domain

Action	Preconditions	Effects
graspA-table(x)	K(*clear*(x)) K(*gripperempty*) K(*ontable*(x)) K(*reachableA*(x)) K(*radius*(x) $\geq$ *minA*) K(*radius*(x) $\leq$ *maxA*)	*add*(K_f, *ingripper*(x)) *add*(K_f, $\neg$*gripperempty*) *add*(K_f, $\neg$*ontable*(x))
findout-open(x)	$\neg K_w$(*open*(x)) K(*ontable*(x))	*add*(K_w, *open*(x))

Actions in PKS are described by their preconditions and effects. An action's preconditions specify the domain properties that must hold for an action to be applied, while an action's effects encode the changes made to the domain properties as a result of executing the action. Table 1 shows two PKS actions from the example domain. Here, K_f refers to a database that models the planner's knowledge of simple facts, while K_w is a specialized database that stores the results of sensing actions that return binary information. An expression like $K(\phi)$ denotes a knowledge-level query that intuitively asks "does the planner know ϕ to be true?"

Given a goal, initial state description, and action list, the planner can build plans that are executable on the robot platform. We currently provide an interface that allows a human user to specify a high-level goal directly to the planning system. Initially, the planner does not know anything about the state of the world. After the robot/vision system performs its early exploration process and begins to produce ISTFs, an initial state description is generated and supplied to the planner automatically with information about newly discovered objects and their sensed properties, described in terms of the high-level fluents. Since PKS can model an agent's incomplete knowledge, the predicate and function instances in the initial state are treated as *known* state information, with all other state information considered to be unknown. We currently assume that the action schema are supplied to the planner as input, as are the mappings from high-level actions to low-level robot motor programs. (In Section 5 we consider how high-level action effects can be learned directly from state information.)

For instance, if we consider the situation in the example domain where two unstacked and open objects *obj1* and *obj2* are on a table, the planner can construct a simple plan using the above domain encoding to achieve the goal of clearing the table:

$$\begin{array}{l}[\textit{graspD-table}(\textit{obj1}),\\ \ \textit{putInto-object}(\textit{obj1}, \textit{obj2}),\\ \ \textit{graspB-table}(\textit{obj2}),\\ \ \textit{putAway}(\textit{obj2})\ ].\end{array} \quad (1)$$

In this plan, *obj1* is grasped from the table and put it into *obj2*, before the stacked objects are grasped and removed to the shelf.

The planner can also build more complex plans by including sensing actions. For instance, if the planner is given the goal of removing the "open" objects from the table, but does not know whether *obj1* is open or not, then it can construct the conditional plan:

$$\begin{array}{l}[\textit{findout-open}(\textit{obj1}),\\ \ \textit{branch}(\textit{open}(\textit{obj1}))\\ \ K^+ :\\ \qquad \textit{graspA-table}(\textit{obj1}),\\ \qquad \textit{putAway}(\textit{obj1})\\ \ K^- :\\ \qquad \textit{nil}\ ].\end{array}$$

This plan senses the truth value of *open*(*obj1*) and reasons about the possible outcomes of this action by including branches in the plan: if *open*(*obj1*) is true (the K^+ branch) then *obj1* is grasped and put away; if *open*(*obj1*) is false (the K^- branch) then no further action is taken.

To execute plans, the planning level interacts with the robot/vision system. Actions are fed to the robot one at a time, where they are converted into motor programs and executed in the world. A stream of ISTFs is also generated, arising from the motor programs being executed. Upon action completion the robot/vision level informs the planner as to any world state changes (the final state of the last ISTF).

An essential component in this architecture is the *plan execution monitor*, which assesses action failure and unexpected state information to control replanning and resensing activities. In particular, the difference between predicted and actual state information is used to decide between (i) continuing the execution of an existing plan, (ii)

asking the vision system to resense a portion of a scene at a higher resolution (in the hope of producing a more detailed state report), and (iii) replanning from the unexpected state using the current state report as a new initial planning state. The plan execution monitor also has the important task of managing the execution of plans with conditional branches, resulting from the inclusion of sensing actions.

When a high-level sensing action is executed at the robot level, the results of the sensing are made available to the robot/vision system in a subsequent ISTF, and passed to the planner as part of a state update. In our example domain, sensing actions like *findout-open* allow the robot to use its lower-level object information to make more informed decisions as to how such actions should best be executed (e.g., for *findout-open* the robot could "poke" an object to determine its openness). The plan execution monitor uses the returned information to decide which branch of a plan it should follow, and feeds the correct sequence of actions to the lower levels. If such information is unavailable, resensing or replanning is triggered as above.

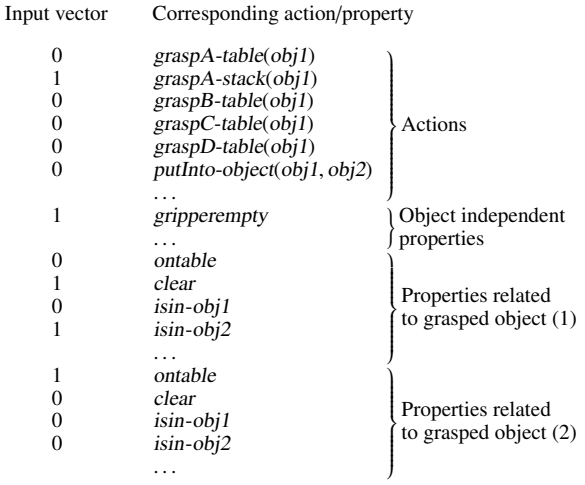


Figure 3. A binary input vector to the learning mechanism

5 LEARNING ACTION REPRESENTATIONS

The planner is capable of constructing plans that direct the robot's actions, in contrast to the reflex-based exploration of the world that the robot must initially perform. This shift from undirected to directed behaviour relies on an action specification that encodes the dynamics of the world in which the robot operates. While we have described how the robot/vision system is capable of generating ISTFs, the state information encoded in such fragments contains information that is both relevant and irrelevant to an action specification. The domain information required for planning actions, however, is more like the information found in a set of OACs, i.e., a generalization of the information in a set of ISTFs. Thus, presented with enough examples of such state transitions, a learning procedure should be able to filter out the irrelevant information and identify the necessary state information required for OACs and planning operators.

Using machine learning techniques to learn action specifications is not a new idea, and prior approaches have addressed this problem using a variety of techniques. For instance, inductive learning [32] and directed experimentation [8] have been applied to data represented in first-order logic, without noise or non-determinism. Other approaches have used schema learning to learn probabilistic action rules operating on discrete-valued sensor data [9]. Also, k-means clustering of equivalence classes, followed by extraction of sensor data features, has been used to train support vector machines (SVMs) to predict deterministic action effects in a given context [4]. [18] proposes a method of modelling actions by learning control laws that change individual perceptual features of the robot's world. Recently, attention has shifted to methods which exploit relational structure in order to improve speed and generalisation performance. [22] generates and refines rules using heuristic search, and shows that relational deictic rules are learnt more effectively than propositional or purely relational rules. [30] uses a logical inference algorithm to efficiently learn rules in relational environments.

Our approach is based on a connectionist machine learning model, namely *kernel perceptron learning* [2, 6]. This approach is particularly useful for our task since it can be shown to provide good performance, both in terms of training time and the quality of the models that are learnt, making it an attractive choice for practical systems.

Learning the complete dynamics of a planning domain requires the ability to learn both action preconditions and effects. Currently, our learning mechanism only addresses the problem of learning action effects, and the action schema and preconditions are supplied as input. Since an action's effects determine the changes made to a state during execution, the problem reduces to learning particular mappings between states. Furthermore, our current mechanism can only learn standard STRIPS and ADL action effects, and is restricted to relational state properties (i.e., no sensing actions or functions).

The input to the learning mechanism uses a vector representation that encodes a description of the action being performed and the state at which the action is applied. For each available action the vector includes an element that is set to 1 if the action is to be performed, or 0 otherwise. For states, we consider object-independent and object-dependent properties separately. In the case of object-independent properties (e.g., *gripperempty*), the vector includes an element for each property, representing its truth value at the state being considered (1 = true, 0 = false). For object-dependent properties we consider each property on a per object basis, and represent only those properties of the objects directly involved in the action being applied, and the objects related in some way to those objects. A form of deictic representation is used (similar to [22]), where objects are specified in terms of their roles in the action, or their roles in a property. Instead of maintaining a "slot" in the input vector for each possible role, roles are allowed to overlap. Thus, each object is represented by a set of inputs, one for each object-specific predicate (e.g., *ingripper*), and each relation with another object (e.g., *isin*). To bind relations to the correct objects, extra predicates are used *isin-obj1*, *isin-obj2*, etc.). This representation significantly reduces the number of inputs since its size is dependent on the actions and relations between objects, rather than the absolute number of objects in the world.

Overall, the input vector has the form: ⟨*actions, object-independent properties, object slot 1 predicates, object slot 2 predicates, ..., object slot n predicates*⟩. Figure 3 shows one such input vector for an action-state pair. In this case, the action performed is *graspA-stack*. The "grasped object" properties are represented in the object *obj1* slot, while the "object below the grasped object" properties are represented in the object *obj2* slot. Here, *gripperempty*, *clear*(*obj1*), *isin*(*obj1*, *obj2*) and *ontable*(*obj2*) are true in the state, since the corresponding bits are set to 1; all other bits are set to 0.

The output of the learning mechanism is a prediction of the properties that will change when the action is performed. The output is also encoded as a binary vector, with each bit representing one property of the state: the output value is 1 if the property changes and 0 if it does not. As with the input vector, object-independent properties

are represented by single elements, and object-specific properties are represented on a per-object basis in slots. Overall, the output vector has the form: ⟨*object-independent properties, object slot 1 predicates, object slot 2 predicates, . . . , object slot n predicates*⟩.

Using the above representation, the learning mechanism is tasked with finding the association between action-precondition pairs and their effects, i.e., rules of the form $\langle A, Pre_A \rangle \rightarrow Eff_A$. Currently, we have focused on learning the effects of standard STRIPS and ADL planning actions. Thus, all action effects involve either conjunctions of predicates (in the case of STRIPS) or conjunctions of predicates conditioned on other conjunctions of predicates (in the case of ADL). As a result, it is sufficient to learn a rule for each effect predicate separately and we can treat the learning problem as a set of binary classification problems, one for each (conditional) effect predicate.

A classifier that is both simple and fast is the *perceptron* [28]. The perceptron maintains a weight vector $\mathbf{w}$ which is adjusted at each training step. The i-th input vector $\mathbf{x}_i \in \{0,1\}^n$ in a class $y \in \{-1, 1\}$ is classified by the perceptron using the decision function $f(\mathbf{x}_i) = sgn(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle)$. If $f(\mathbf{x}_i)$ is not the correct class then $\mathbf{w}$ is set to $\mathbf{w} + y\mathbf{x}$; if $f(\mathbf{x}_i)$ is correct then $\mathbf{w}$ is left unchanged. Provided the data is linearly separable, the perceptron algorithm is guaranteed to converge on a solution in a finite number of steps [21, 17]. Otherwise, the algorithm oscillates, changing $\mathbf{w}$ at each misclassified input vector.

Since the problem of learning action effects is not linearly separable in general, we adapt the perceptron algorithm by applying the *kernel trick* [6]. By doing so, we implicitly map the input feature space into a higher-dimensional space where the data is linearly separable. Since the mapping is implicit, we avoid a massive expansion in the number of features, which may make the problem computationally infeasible. The kernel trick is applied by rewriting the decision function in terms of the dot product of the input vectors:

$$f(\mathbf{x}_i) = sgn(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle) = sgn(\sum_{j=1}^{n} \alpha_j y_j \langle \mathbf{x}_j \cdot \mathbf{x}_i \rangle),$$

where α_j is the number of times the j-th example has been misclassified by the perceptron. By replacing the dot product with a *kernel function* $k(\mathbf{x}_i, \mathbf{x}_j)$ which calculates $\langle \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) \rangle$ for some mapping ϕ, the perceptron algorithm can be run in higher dimensional space without requiring the mapping to be explicitly calculated. An ideal kernel is one which allows the perceptron algorithm to run over the feature space of all conjunctions of features in the original input space, allowing an accurate representation of the exact conjunction of features (action and preconditions) corresponding to a particular effect. In our case, the kernel $k(x, y) = 2^{same(x,y)}$ is used, where $same(x, y)$ is the number of bits with the same value in both x and y [29, 10]. (See [19] for a more detailed discussion of this approach.)

6 INTEGRATION AND EMPIRICAL RESULTS

In this section we consider two separate interactions between the components described above, forming part of the larger system we are in the process of implementing (see Figure 4). In Section 6.1 we consider the link between the planning level and the robot/vision level, and the execution of high-level plans on the robot platform. In Section 6.2 we focus on the learning mechanism and the actions that arise from the object manipulation scenario. Certain aspects of our system, such as the plan execution monitor and the inclusion of the learning mechanism within the larger system, are currently under development and have not yet been fully implemented. The robot/vision system forming the basis of our implementation consists of an industrial 6 degrees of freedom robot with a two finger

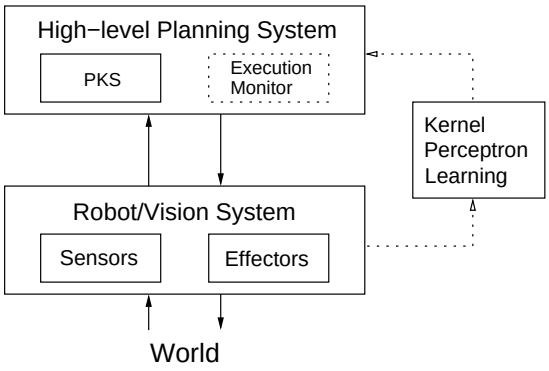


Figure 4. System components and proposed/current interactions

grasper, a high resolution stereo camera system, and a haptic force-torque sensor mounted between the robot and grasper, providing the measurement of forces at the wrist.

6.1 Linking high-level plan generation with robot/vision-level plan execution

From an integration point of view, the robot/vision system is currently linked directly to the planning level and we are experimenting with plan generation and execution. Since the planner is not able to handle raw sensor data as a state description, the low-level ISTFs generated by the robot/vision system must be abstracted into a language that is understandable by the planner. As a result, sensor data is "wrapped" and reported to the planner in the form of "symbolic" ISTFs with state representations that include predicates and functions. Since our present focus is on object and action learning, we have simply hard-coded the mappings between certain sensor combinations and the corresponding high-level properties.

For instance, some of the predicates used in the example manipulation domain are computed as follows:

- *ingripper*, *gripperempty*: Initially the gripper is empty and the predicate *gripperempty* is formed. As soon as the robot grasps an object, and confirms that the grasp is successful by means of the gripper not closing up to mechanical limits, the system knows that it has the object in its hand and can form a predicate *ingripper*(*objX*), using its visual information about discovered objects to identify the label *objX* corresponding to the object in the gripper. A negated predicate ¬*gripperempty* is also generated, as are negated *ingripper* instances for objects not in the gripper. Releasing the object returns the gripper to an empty state again.
- *reachableX*: Based on the position of a circle forming the top of a cylindrical object in the scene, as returned by the circle detection algorithm, we can compute possible grasp positions (for the different grasp types) for each object. Using standard robotics path planning methods we can then compute if there is a collision-free path between the start position and the pose the gripper needs to reach the object for a particular grasp.
- *isin*, *clear*, *instack*: These three predicates are computed based on geometric reasoning. Since the object height is not known we can only use the x, y-plane information. Furthermore the fact that objects with a bigger radius are lower in the stack is assumed. Objects whose centres (in the x, y-plane) are closer than 40mm are selected as stack candidates. The sorted stack candidates can then be checked for real inclusion using the circle centres and radii.

- *open*: We do not assume that all objects in the world are "open." Unlike the previous properties which can be determined directly from ordinary sensor data, the robot must first perform an explicit test in order to determine an object's openness. In this case, the robot gripper is used to "poke" inside the potential opening of an object. If the robot encounters a collision where forces acting upon the gripper are above a certain threshold, then the object is assumed to be closed. Otherwise, we assume the object is open. (We also envision a second, purely visual test for openness using dense stereo, but this approach has not yet been implemented.)

After an initial exploration of its environment, the robot/vision system provides the planner with a report of the current set of objects it believes to be in the world, along with a (possibly partial) state report of the sensed properties of those objects. Using this information as its initial knowledge state, and the high-level action specification described in Section 4, the planner attempts to construct a plan to achieve the goal of clearing a set of objects from the table.

Once a plan has been generated, it is passed to the plan execution monitor which sends actions to the robot/vision level one step at a time. At the robot level, a high-level action is decomposed into a set of motor programs which are then executed by the robot in the world. Currently, the mapping of actions to motor programs is preprogrammed and supplied as part of the input to the system. During the execution of low-level motor programs, a stream of ISTFs is generated and recorded by the robot/vision system. After an action has been executed its success or failure is reported back to the plan execution monitor, along with a new report on the state of the world (the final state of the last ISTF). In our current implementation, the plan execution monitor simply terminates the execution of a plan if it encounters an unexpected state property, or a reported failure of an action. Otherwise, it sends the next action to the robot for execution. (No replanning or focused resensing is performed.) For instance, Figure 5 shows the robot executing the four-step plan described in (1) of Section 4 for clearing the table. (The "shelf" in this case is a special area at the side of the table.)

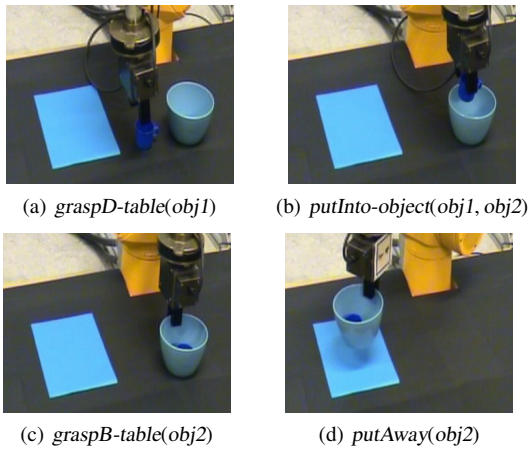
When a conditional plan with sensing actions is executed, the plan execution monitor sends *findout-open* actions to the robot/vision level like any other action. At the robot level, such an action is executed as the specific "poke" test described above to determine an object's openness. The results of this test are returned to the plan execution monitor as part of the updated state report. The monitor then uses this information to determine which branch of the conditional plan it should follow. From the point of view of the robot, it will only receive a sequential stream of actions and will be unaware of the conditional nature of the plan being executed. Figure 6 shows the robot testing the openness of two objects after receiving a sensing action from the planning level. In (a), the test fails since the object is not open; in (b) the test succeeds and the object is assumed to be open.

6.2 Learning STRIPS and ADL action effects in the object manipulation domain

Separate from the above robot/vision-planner integration, we established a preliminary link between the action effect learning mechanism and the planner. In particular, we applied our learning procedure to learn the effects of STRIPS and ADL planning actions, using data simulated from the example object manipulation domain.

The learning mechanism was evaluated using data similar to the ISTFs the robot/vision system is capable of producing. Both STRIPS and ADL versions of the high-level actions were considered. (For example, the two actions *graspA-stack* and *graspA-table* described in Section 4 were merged into a single ADL action, along with other changes.) Sensing actions and references to functional fluents were ignored. Two data sets were constructed to train and test the learning mechanism. Individual input vector instances were generated by randomly selecting an action, and setting the inputs for the preconditions required for the action to 1. The action input was set to 1, and all other action inputs to 0. The remaining input bits were used to create the two separate data sets. For the training data, half of the inputs in each instance were randomly set to 0 or 1, with the other half all set to 0 (vice versa for the testing data). Outputs were set to 1 if a state property changed as a result of the action and 0 if not. Thus, the data used to train the learning mechanism incorporated the (strong) assumptions that (i) all the necessary precondition information for an associated action was included as part of an input vector, and (ii) no spurious state changes was represented as part of an output vector. Noise was introduced in the irrelevant bits of the input vector, however, only relevant changes were included in the corresponding output vector.

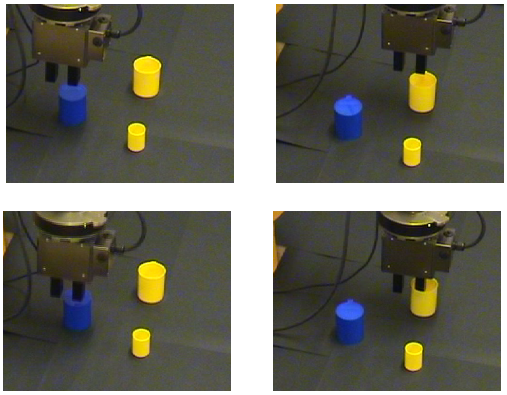
The learning mechanism was evaluated over multiple test runs using 3000 training and 500 testing examples. To determine an error bound on our results, 10 runs with different randomly generated training and testing sets were used. (All testing was done on a 2.4 GHz quad-core system with 6 Gb of RAM. All times were measured for Matlab 7.2.0.294.) The results of our testing are shown in Fig-



(a) *graspD-table*(*obj1*) (b) *putInto-object*(*obj1*, *obj2*)

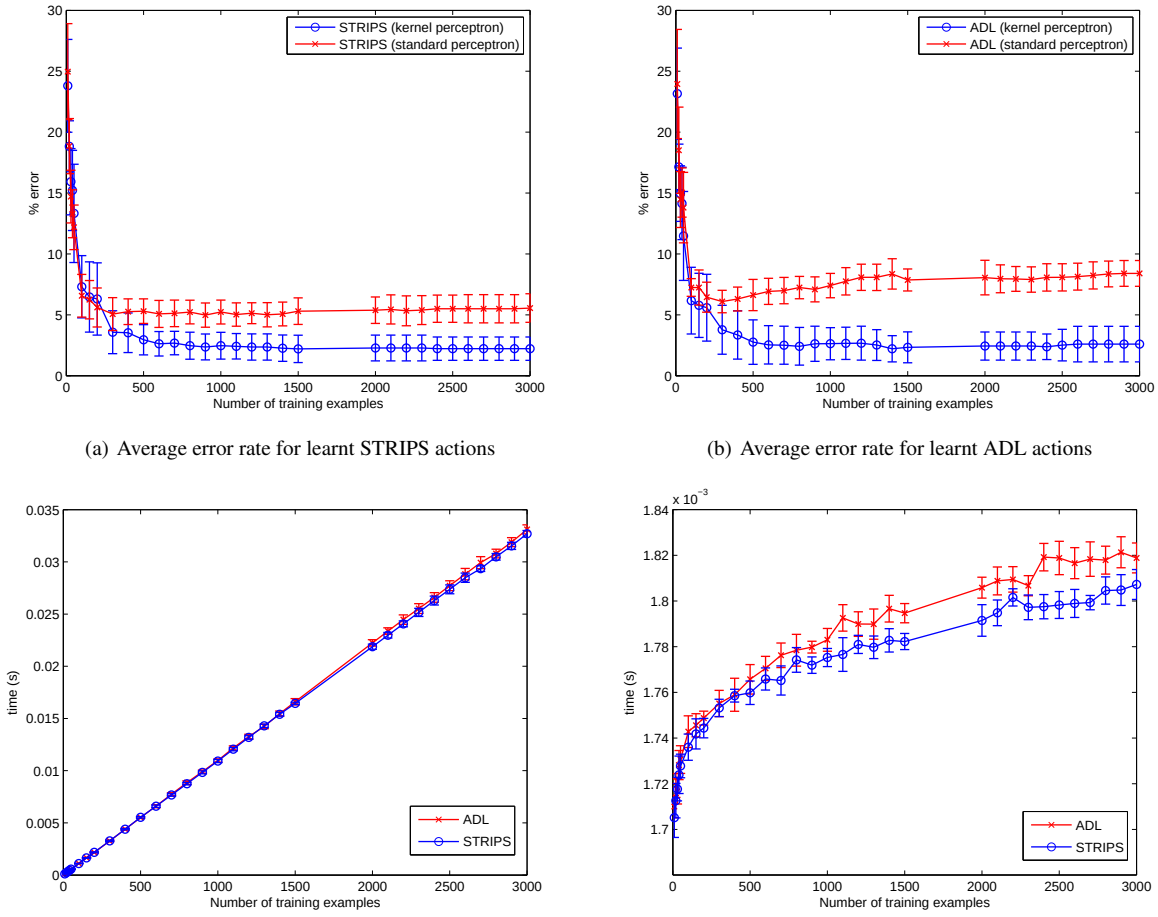
(c) *graspB-table*(*obj2*) (d) *putAway*(*obj2*)

Figure 5. Executing a high-level plan to clear a table



(a) Object is not open (b) Object is open

Figure 6. Testing the openness of an object



(a) Average error rate for learnt STRIPS actions

(b) Average error rate for learnt ADL actions

(c) Training time for STRIPS and ADL actions

(d) Prediction time for STRIPS and ADL actions

Figure 7. Results from experiments in the object manipulation domain (from [19])

ure 7. (A more detailed analysis of these results and description of our implementation can be found in [19].) In (a), the error rate for the learnt STRIPS actions is shown, while (b) shows the error rate for the ADL actions. In both cases, the average error dropped to less than 3% after 700 training examples. The standard perceptron error rate, included for comparison, shows significantly worse performance: over 5% error after 3000 training examples. In (c), the training time for both STRIPS and ADL actions is shown (for 1 bit of the effect vector), while (d) shows the prediction time (for 1 bit of 1 prediction).

In practical terms, the learning mechanism was quite efficient, requiring 0.035 seconds to train the system on 3000 examples and 1.84 $\times 10^{-3}$ seconds to test the system per output. For our particular example domain, there was little difference between the training and prediction times of STRIPS actions, compared with those for ADL actions. (In general we expect performance on ADL domains will always take longer than STRIPS domains, particularly when the conditional effect training examples are very dissimilar to the other training examples available.) Overall, the learning mechanism was able to effectively abstract away the irrelevant information from the ISTFs to produce a high-quality model of the action effects suitable for planning (at least for our current example domain).

7 DISCUSSION

From a representational point of view we have argued that ISTFs and OACs, grounded from actions performed at the robot level, can be viewed as the representational unit that underlies higher-level representations of objects, properties, and actions ("representation through integration"). As the low-level robot/vision system explores the world, successful actions produce ISTFs; on the basis of multiple experiences of particular ISTFs, OACs and high-level action models can be learned. Although some aspects of our approach are currently hard-coded (e.g., the action/motor program mappings), our learning mechanisms are nevertheless able to abstract away from "irrelevant" state information in the ISTFs to learn certain high-level OAC relationships from the robot's interactions with the world.

The resulting representations also enable interesting interactions between the components of the system ("integration through representation"). For instance, the planner can ignore some details about the execution of actions at the robot level (e.g., sensing actions like *findout-open*) and can avoid making certain commitments that are better left to the robot level (e.g., planning-level grasping actions are unaware of low-level properties like object location, gripper orientation relative to an object, etc.). Thus, we do not try to control all

aspects of robot behaviour at the planning level, but apply the planner's strengths to problems it can more readily solve. (For instance, PKS does not perform path planning but is more proficient at planning information-gathering operations.) As future work we are extending these ideas, for instance to allow the robot/vision system to choose between a set of possible tests to perform when executing a *findout-open* sensing action, while leaving the planning-level action specification unchanged.

We have focused on two particular learning problems in this work: object learning and action effect learning. As a result, we have avoided addressing other learning problems (e.g., learning the low-level sensor combinations that lead to particular high-level properties, or the mapping of high-level actions to low-level motor programs), which we leave to future work. Our focus on an implementation "from the world level to the knowledge level," however, provides us with a suitable testing framework for investigating such learning challenges as well as new planning contexts. Moreover, we are also interested in using this platform to explore other high-level learning tasks such as language acquisition.

We must also improve the scalability of our approach and overcome certain assumptions that are not realistic in real-world robotic systems. For instance, the learning mechanism has mainly been tested using state descriptions that are more "complete" than the ISTFs the robot/vision system is likely to produce. One way we can adapt our approach is by using a noise-tolerant variant of the perceptron algorithm, such as adding a margin term [11]. We also believe these techniques can be applied to irrelevant output data (i.e., irrelevant state changes in the action effects), since such changes behave like noise. Additional work is needed to extend our approach to more complex action representations, notably sensing actions and functions. We also believe our approach can be extended to learn action preconditions, provided it is possible to only represent a small number of objects in the state at a time. An attentional mechanism of some sort may be of help in this task [14]. Finally, although we have tested our learning mechanism on simulated data from the same domain used for the robot/vision-planner experiments, we are also aiming to test our learning mechanism with online data generated directly from the robot/vision system. Additional work is also needed to complete the remaining components of our system, most notably the plan execution monitor.

Our approach for integrating a robot/vision system with a high-level planner and action learning mechanism combines ideas from robot vision, symbolic knowledge representation and planning, and connectionist machine learning. The current state of our work highlights some significant interactions between the specific components of our system, however, we believe our approach is much more general and can be applied to other robot platforms and planners. (For instance, we have recently begun work to test some of our components and specifications on a humanoid robot platform.) The components we describe in this paper form part of a larger project called PACO-PLUS[4] investigating perception, action, and cognition—combining robot platforms with high-level representation and reasoning based on formal models of knowledge and action [12].

ACKNOWLEDGEMENTS

The work in this paper was partially funded by the European Commission through the PACO-PLUS project (FP6-2004-IST-4-27657).

[4] See `www.paco-plus.org` for more information about this project.

REFERENCES

[1] D. Aarno, J. Sommerfeld, D. Kragic, N. Pugeault, S. Kalkan, F. Wörgötter, D. Kraft, and N. Krüger, 'Early reactive grasping with second order 3D feature relations', in *Recent Progress in Robotics; Viable Robotic Service to Human, Selected papers from ICAR'07*, eds., S. Lee, I. Hong Suh, and M. Sang Kim, LNCIS Series, Springer-Verlag, (2007).

[2] M. A. Aizerman, E. M. Braverman, and L. I. Rozoner, 'Theoretical foundations of the potential function method in pattern recognition learning', *Automation and Remote Control*, **25**, 821–837, (1964).

[3] Renaud Detry, Nicolas Pugeault, and Justus H. Piater, 'Probabilistic pose recovery using learned hierarchical object models', in *International Cognitive Vision Workshop (Workshop at the 6th International Conference on Vision Systems)*, (2008).

[4] Mehmet R. Doğar, Maya Çakmak, Emre Uğur, and Erol Şahin, 'From primitive behaviors to goal directed behavior using affordances', in *Proc. of Intelligent Robots and Systems (IROS 2007)*, (2007).

[5] Richard E. Fikes and Nils J. Nilsson, 'STRIPS: A new approach to the application of theorem proving to problem solving', *Artificial Intelligence*, **2**, 189–208, (1971).

[6] Yoav Freund and Robert Schapire, 'Large margin classification using the perceptron algorithm', *Machine Learning*, **37**, 277–296, (1999).

[7] Christopher Geib, Kira Mourão, Ron Petrick, Nico Pugeault, Mark Steedman, Norbert Krueger, and Florentin Wörgötter, 'Object action complexes as an interface for planning and robot control', in *IEEE-RAS Humanoids-06 Workshop: Towards Cognitive Humanoid Robots*, (2006).

[8] Yolanda Gil, 'Learning by experimentation: Incremental refinement of incomplete planning domains', in *Proc. of ICML-94*. MIT Press, (1994).

[9] Michael Holmes and Charles Isbell, 'Schema learning: Experience-based construction of predictive action models', in *Advances in Neural Information Processing Systems (NIPS) 17*, pp. 585–562, (2005).

[10] Roni Khardon, Dan Roth, and Rocco A. Servedio, 'Efficiency versus convergence of boolean kernels for on-line learning algorithms', *Journal of Artificial Intelligence Research*, **24**, 341–356, (2005).

[11] Roni Khardon and Gabriel M. Wachman, 'Noise tolerant variants of the perceptron algorithm', *Journal of Machine Learning Research*, **8**, 227–248, (2007).

[12] D. Kraft, E. Başeski, M. Popović, A. M. Batog, A. Kjær-Nielsen, N. Krüger, R. Petrick, C. Geib, N. Pugeault, M. Steedman, T. Asfour, R. Dillmann, S. Kalkan, F. Wörgötter, B. Hommel, R. Detry, and J. Piater, 'Exploration and planning in a three-level cognitive architecture', in *Int. Conference on Cognitive Systems (CogSys 2008)*, (2008).

[13] D. Kraft, N. Pugeault, E. Başeski, M. Popović, D. Kragic, S. Kalkan, F. Wörgötter, and N. Krüger, 'Birth of the Object: Detection of objectness and extraction of object shape through object action complexes', *Special Issue on "Cognitive Humanoid Robots" of the International Journal of Humanoid Robotics*, (2008). (accepted).

[14] Danica Kragic, Måarten Björkman, Henrik I. Christensen, and Jan-Olof Eklundh, 'Vision for robotic object manipulation in domestic settings', *Robotics and Autonomous Systems*, **52**(1), 85–100, (July 2005).

[15] N. Krüger, M. Van Hulle, and F. Wörgötter, 'ECOVISION: Challenges in early-cognitive vision', *Int. Journal of Computer Vision*, (2006).

[16] John McCarthy and Patrick J. Hayes, 'Some philosophical problems from the standpoint of artificial intelligence', *Machine Intelligence*, **4**, 463–502, (1969).

[17] Marvin L. Minsky and Seymour A. Papert, *Perceptrons*, The MIT Press, December 1969.

[18] Joseph Modayil and Benjamin Kuipers, 'Where do actions come from? Autonomous robot learning of objects and actions', in *AAAI Spring Symposium Series*, (2007).

[19] Kira Mourão, Ronald P. A. Petrick, and Mark Steedman, 'Using kernel perceptrons to learn action effects for planning', in *International Conference on Cognitive Systems (CogSys 2008)*, (2008).

[20] R.M. Murray, Z. Li, and S.S. Sastry, *A mathematical introduction to Robotic Manipulation*, CRC Press, 1994.

[21] Albert B. Novikoff, 'On convergence proofs for perceptrons', in *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume 12, pp. 615–622, (1963).

[22] Hanna M. Pasula, Luke S. Zettlemoyer, and Leslie P. Kaelbling, 'Learning symbolic models of stochastic domains', *JAIR*, **29**, 309–352, (2007).

[23] Edwin P. D. Pednault, 'ADL: Exploring the middle ground between STRIPS and the situation calculus', in *Proc. of KR-89*, pp. 324–332. Morgan Kaufmann, (1989).
[24] Ronald P. A. Petrick and Fahiem Bacchus, 'A knowledge-based approach to planning with incomplete information and sensing', in *Proc. of AIPS-2002*, pp. 212–221. AAAI Press, (2002).
[25] Ronald P. A. Petrick and Fahiem Bacchus, 'Extending the knowledge-based approach to planning with incomplete information and sensing', in *Proc. of ICAPS-04*, pp. 2–11. AAAI Press, (2004).
[26] N. Pugeault, F. Wörgötter, and N. Krüger, 'Multi-modal scene reconstruction using perceptual grouping constraints', in *Proc. of the IEEE Computer Society Workshop on Perceptual Organization in Computer Vision*, (2006).
[27] Raymond Reiter, *Knowledge In Action: Logical Foundations for Specifying and Implementing Dynamical Systems*, MIT Press, 2001.
[28] Frank Rosenblatt, 'The perceptron: a probabilistic model for information storage and organization in the brain', *Psychological Review*, **65**(6), 386–408, (November 1958).
[29] Ken Sadohara, 'Learning of boolean functions using support vector machines', in *Proc. of Algorithmic Learning Theory,* Lecture Notes in Artificial Intelligence, volume 2225, pp. 106–118. Springer, (2001).
[30] Dafna Shahaf and Eyal Amir, 'Learning partially observable action schemas', in *Proc. of AAAI-06*. AAAI Press, (2006).
[31] Mark Steedman, 'Plans, affordances, and combinatory grammar', *Linguistics and Philosophy*, **25**, 723–753, (2002).
[32] Xuemei Wang, 'Learning by observation and practice: An incremental approach for planning operator acquisition', in *Proc. of ICML-95*, pp. 549–557, (1995).